EXHIBIT D

Implementing TCP/IP communications with HyperCard

Introduction

This article describes how to implement TCP/IP communications with HyperCard in three steps. First, it briefly examines the tools used to access information resources available through the Internet. Second, it outlines the necessary hardware and software requirements to make TCP/IP communications happen on a Macintosh. Third, it illustrates the implementation process with two stacks: Mini-Atlas and ListManager.

Note that these scripts are intended for intermediate and advanced HyperTalk scripters. This article assumes the reader has at least a working knowledge of HyperTalk and has scripted a couple of stacks.

The Internet

The Internet is a vast network of computers. The set of communications standards making the Internet possible is called the Transmission Control Protocol/Internet Protocol or TCP/IP for short. With these standards you can use telnet to log into remote computers, transfer files from remote sites with FTP, and send electronic mail to remote computers. If you use a Macintosh computer and it is directly connected to the Internet, then you probably use programs like BYU/NCSA Telnet and TN3270, XferIt, or an electronic mail gateway to remote computers. These programs are general purpose tools for remote access; they are not intended to provide you with access to specific information services. With HyperCard it is possible to create specialized information gathering tools to access specialized information services.

Requirements

The necessary hardware and software requirements to implement TCP/IP communications from within HyperCard include:

- a Macintosh with a direct connection to the Internet,
- any version of HyperCard,
- MacTCP, and
- the XCMDs from the HyperCard TCP Toolkit.

The most difficult of these components to acquire is the direct connection to the Internet. (If you are uncertain whether or not you have a direct connection to the Internet, then talk to your systems administrator.)

HyperCard comes with every Macintosh. MacTCP is an operating system extension enabling the Macintosh to implement the TCP/IP protocols. It is available from the Apple Developers Association (APDA) (1-800-282-2732).

Lastly, the HyperCard TCP Toolkit is a set of XCMDs which calls the routines within MacTCP. XCMDs are compiled pieces of programming code usually written in a the C or Pascal programming language. XCMDS can be added to your HyperCard stacks. In turn, they add additional functionality to the HyperCard HyperTalk language. The TCP Toolkit is available from APDA, America Online, and from a number of anonymous FTP sites as well.(1)

The XCMDs described

The following text was taken from the introductory card of the HyperCard TCPToolkit.

> The HyperCard TCP Toolkit consists of a set of HyperTalk commands and functions which allow HyperCard stacks to establish TCP connections and send data across them. There is also a function for performing name to address translation.
>
> A connection is established using the TCPActiveOpen function, which establishes a connection with the remote socket [a connection between computer processes allowing them to communicate in a fast, reliable manner] and returns a connection ID. This connection ID is used to specify which connection to send and

receive data on for the TCPCharsAvailable, TCPSend, TCPRecvChars, TCPRecvUpTo, and TCPRecvMsg commands and functions. The TCPState function returns the current state of the connection. To gracefully close a connection, the TCPClose command is called; the HyperTalk script should then wait for the connection to close, by calling TCPStatus until it returns "closed", and then calling TCPRelease. (A call to TCPRelease, without first closing the connection, will abort the connection.)

Alternatively, TCPPassiveOpen will allow connection to be accepted on a particular socket. The TCPState function can then be called to determine when the connection is established.

Here is a simple algorithm for establishing a connection, performing a dialog with the remote computer, and closing a connection.

1. Resolve the remote computer's name or address with TCPNameToAddr.
2. Open a connection to the remote computer with TCPActiveOpen.
3. Make sure a connection is established with TCPState.
4. Read incoming characters with TCPRecvUpTo.
5. Send outgoing characters with TCPSend.
6. If the dialog is not complete, then go to 3.
7. Close the connection with TCPClose.
8. Release the connection with TCPRelease.

Using this simple algorithm I have created two information gathering tools: Mini-Atlas and ListManager. The following text describes these tools.

Mini-Atlas

Mini-Atlas is a client for the Geographic Name Server. The Geographic Name Server contains brief information about a vast majority of United States cities and geographic landmarks. This sort of information usually includes:

- the name of the place,
- the type of place (city, river, lake, mountain, et cetera)
- latitude and longitude,
- a remark,
- the county it is located in,
- the telephone area code,
- zip codes, and
- 1980 census population.

To manually query the Geographic Name Server you would follow these steps:

1. Open a TCP/IP connection to martini.eecs.umich.edu on port 3000.
2. Wait for the period (.) prompt.
3. Send a query in the form of postal-like address, for example, Lancaster PA, Lake Erie, or Los Angeles.
4. Wait for the period prompt.
5. Logoff.

Notice the similarities between manually querying the Geographic Name Server and the basic query algorithm listed in the section above. Querying the Geographic Name Server for "lancaster, pa" yields the following output during the Mini-Atlas session:

```
# Geographic Name Server, Copyright 1989, 1990 Merit Inc.
# All rights reserved.
# >>> NEW VERSION!!! <<< Use "help" or "?" for details.
.
lancaster, pa
0 Lancaster
1 42071 Lancaster
2 PA Pennsylvania
3 US United States
R county seat
F 45 Populated place
```

```
L 40 02 16 N  76 18 21 W
P 54725
E 357
Z 17600
Z 17601
Z 17602
Z 17603
Z 17604
Z 17605

.
bye
```

What the HyperCard script in Mini-Atlas does is automate the above manual communications and query processes so that by answering a one simple question the user can retrieve data from the Geographic Name Server without having to manually initiate the communications procedures. All the necessary HyperTalk code for MiniAtlas is listed in the Appendix. The Mini-Atlas souce code is available at http://infomotions.com/musings/tcp-communications/Mini-Atlas.sea.hqx. What follows is descriptions of essential parts of the code.

First, the Geographic Name Server's IP name is resolved.(2) (Step 1.)

```
put item 1 of TCPNameToAddr("martini.eecs.umich.edu",,) into theIPAddress
```

If an error did not occur, then a connection is opened with theIPAddress on port 3000.(3) (Step 2.)

```
put TCPActiveOpen(theIPAddress, 3000, 0) into connectionID
```

Again, if no error occurred, then wait for the period prompt with a handler called WaitForPeriod. The heart of this function waits for a linefeed character (ASCII character 10) and returns the character and the string preceding it. (Steps 3. and 4.)

```
put TCPRecvUpTo(connectionID, linefeed, 60, empty) into theResponse
```

Once theResponse is retrieved, simple examination determines whether or not it is the period prompt. If it is the period prompt, then our query is retrieved from the user's input and sent with the SendLine handler. SendLine verifies the connection and sends theText (in this case, the query). (Step 5.)

```
if TCPState (connectionID) is "established" then
  TCPSend connectionID, theText & return & linefeed
end if
```

After sending the query, WaitForPeriod is called again. Not only does this handler wait for the period prompt, it also puts all incoming text into a field for analysis later. After the period prompt is sent from the server, the connection is closed and released with the Disconnect handler where it calls the following two commands. (Steps 7. and 8.)

```
TCPClose connectionID TCPRelease connectionID
```

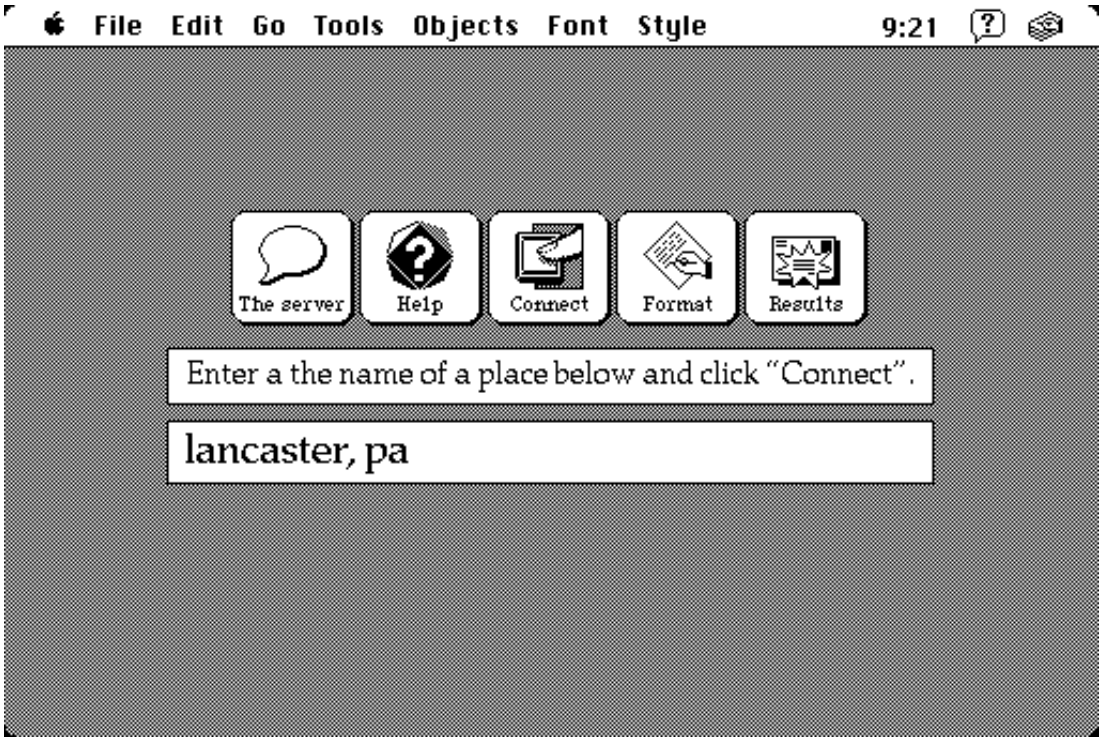

Figure 1. The opening screen of the Mini-Atlas simply requires the user to enter a query for the Geographic Name Server. After clicking the "Connect" button the Mini-Atlas opens a connection to the Geographic Name Server, downloads the results, and closes the TCP/IP connection.

The most difficult aspect of developing this information tool is creating a way to analyze the retrieved data. I have parsed my retrieved data into records and created new cards for each record. Each card contains four buttons and one field. The field contains the data in human-readable form. Three of the buttons are used for navigation. The last button reads the place's latitude and longitude, converts them into screen coordinates, and literally pinpoints the place on a world map. Other things could be done with the data. For example, many place names could be input and the distances between them calculated. Populations could be compared. The Mini-Atlas could be used to answer simple reference questions about a place.

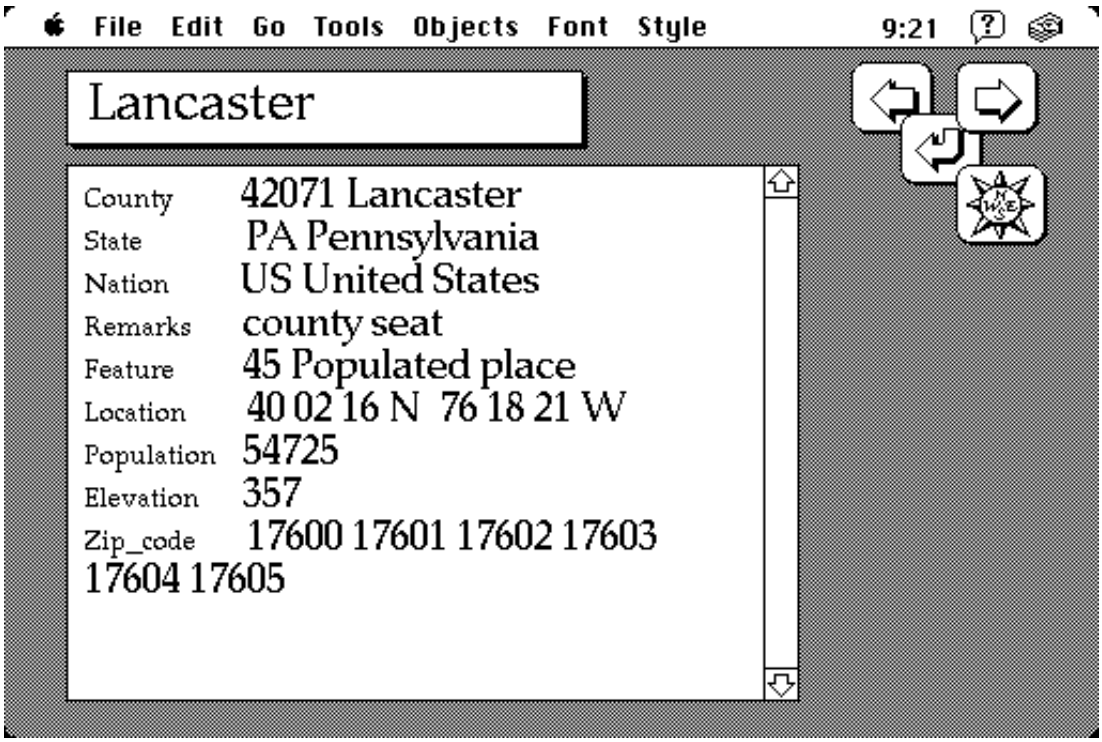

Figure 2. After the connection to the Geographic Name Server is closed, the data is formated into human-readable form.

Figure 3. Lastly, the user has the option of clicking the "Map" button. Consequently, the query's latitude and longitude (location) are translated into screen coordinates and literally pinpointed on a world map.

ListManager

Another, more interesting application is the ListManager, a front-end to LISTSERV programs which operate electronic lists such as PACS-L, AUTOCAT, and LIBREF-L. ListManager automates the procedures necessary to search the archives of these lists by keyword Boolean queries, to temporarily turn off your mail from the list, to retrieve a list of participants of the list, or to retrieve files from the lists. With the ListManager you can do all these things and more simply by answering a few questions and clicking a few buttons. In other words, the ListManager conducts a simple reference interview querying the user about LISTSERV needs. In the process, it creates a Simple Mail Transfer Protocol (SMTP) message. Using the XCMDs from the HyperCard TCP Toolkit, the ListManager sends the message to the user's mail server. In turn, the mail server forwards the message to the LISTSERV program where it processes it and sends a reply to the user's email address. The source code for the ListManager is available at http://infomotions.com/musings/tcp-communications/ListManager.sea.hqx. There are two parts to the ListManager: the reference interview and the transmission of the resulting email message. The reference interview process asks you questions. Based on the answers to those questions, it asks other questions. When the question and answer process is complete, the result is an SMTP message. The first question is "What is your name?" Next it asks for your email address and the name of your mail server. It then asks you, "To what list do you want to send mail?" Then ListManager asks, "Do you want to send mail to the server or to the participants of the list?" If you choose "participants" then you are given the opportunity to write a posting. On the other hand, if you choose "server" then you are presented with the various commands that can be sent to the server like: subscribe, unsubscribe, index, get, and review.

There are two parts to the ListManager: the reference interview and the transmission of the resulting email message.

The reference interview process asks you questions. Based on the answers to those questions, it asks other questions. When the question-and-answer process is complete, the result is an SMTP message. The first question is "What is your name?" Next it asks for your email address and the name of your mail server. It then asks you, "To what list do you want to send mail?" Then ListManager asks, "Do you want to send mail to the server or to the participants of the list?" If you choose "participants" then you are given the opportunity to write a posting. On the other hand, if you choose "server" then you are presented with the various commands that can be sent to the server like: subscribe, unsubscribe, index, get, and review.

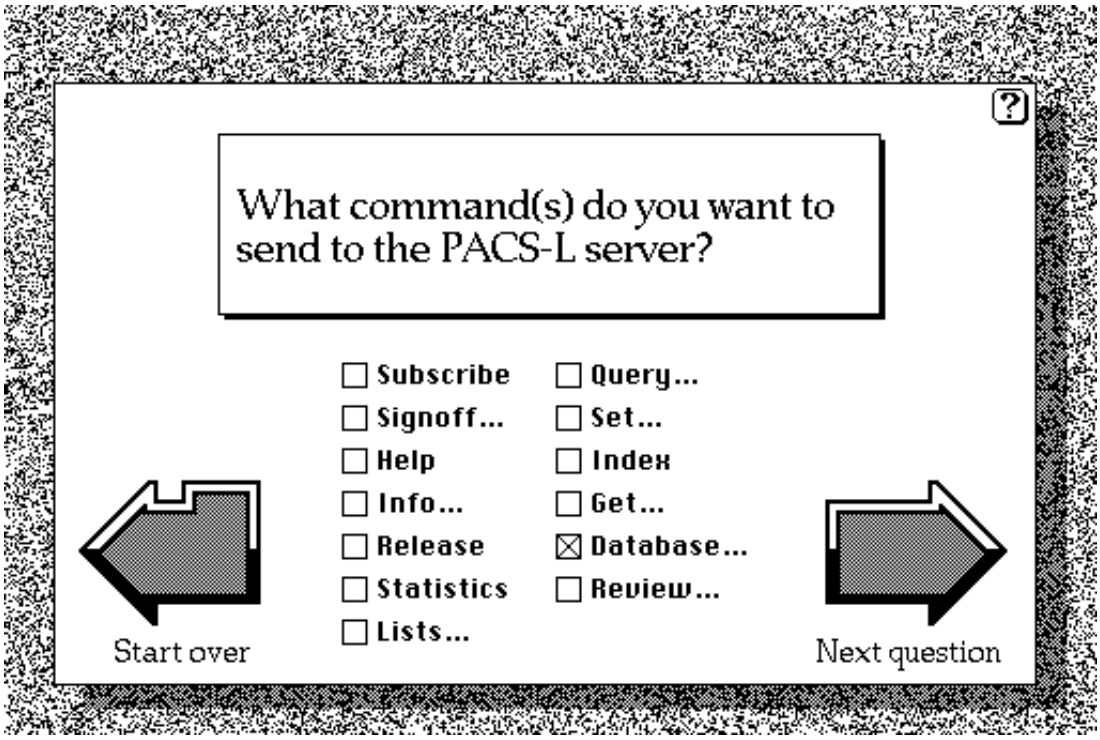

Figure 4. The ListManager implements a simple reference interview. During the interview the operator is asked, "What command(s) do you want to send the PACS-L [selected] server?" Information about the commands can be retrieved by clicking the question mark button.

Here are some example results of the reference interview process. If I wanted to subscribe to PACS-L, then the resulting email message would look like this:

```
HELO ericmorgan.lib.ncsu.edu
MAIL FROM:<eric_morgan@ncsu.edu>
RCPT TO:<listserv@UHUPVM1.bitnet>
DATA
FROM:<eric_morgan@ncsu.edu>
TO:<listserv@UHUPVM1.bitnet>

subscribe PACS-L Eric Morgan
.
QUIT
```

On the other hand, if I wanted to retrieve a list of all the files available from PACS-L, using my email address as an example, then the resulting email message would look like:

```
HELO ericmorgan.lib.ncsu.edu
MAIL FROM:<eric_morgan@ncsu.edu>
RCPT TO:<listserv@UHUPVM1.bitnet>
DATA
FROM:<eric_morgan@ncsu.edu>
TO:<listserv@UHUPVM1.bitnet>

index PACS-L
.
QUIT
```

Then, if one wanted to query the PACS-L list for messages (records) containing the terms "ALA" and "preconference" posted between May 1, 1990 and August 31, 1990 where the subject line sounds like "shikago" (4) and I wanted the results sent in a user-defined author-subject format, then the resulting email message would look like:

```
HELO ericmorgan.lib.ncsu.edu
```

```
MAIL FROM:<eric_morgan@ncsu.edu>
RCPT TO:<listserv@UHUPVM1.bitnet>
DATA
FROM:<eric_morgan@ncsu.edu>
TO:<listserv@UHUPVM1.bitnet>

//   JOB  Echo=No
Database Search DD=Rules
//Rules DD   *
S ALA preconference in PACS-L where subject sounds like shikago from 90/1/5 -
to 90/31/8
F AS: #.6R0 "Record" From.25 "Author" Subject.80 "Subject"
I AS
/*
.
QUIT
```

The output of a search similar to the one above looks like this:

```
> S 'ALA' preconference in PACS-L
--> Database PACS-L, 7 hits.

> F AS: #.6R0 "Record" From.25 "Author" Subject.80 "Subject"

> I AS
Record Author                    Subject
------ ------                    -------
001594 UJMB00@SDNET.BITNET       ILL, Copyright and Internet Catalogs
003197 KATHY@BAKER.DARTMOUTH.EDU LITA Human-Machine Interface Interest Group pl+
003468 BR.WCC@RLG.BITNET         Screen design: since you asked
003540 FCLMID@NERVM.BITNET       LITA Screen Design Preconference
003654 LIBPACS@UHUPVM1.BITNET    Screen Design
005747 BR.WCC@RLG.BITNET         Screen Design/Online Catalog project status re+
006113 CHIPOKR@elmer1.bobst.nyu.+Images in the OPAC
```

Finally, if I wanted to retrieve record numbers 001594, 003468, 006113 of the resulting set from the search above, then the resulting email message would be:

```
HELO ericmorgan.lib.ncsu.edu
MAIL FROM:<eric_morgan@ncsu.edu>
RCPT TO:<listserv@UHUPVM1.bitnet>
DATA
FROM:<eric_morgan@ncsu.edu>
TO:<listserv@UHUPVM1.bitnet>

//   JOB  Echo=No
Database Search DD=Rules
//Rules DD   *
search * in PACS-L
print all of 006113 006113 006113
/*
.
QUIT
```

Granted, the first 7 and last 2 lines of each of these examples could be created by me email program, but everything in between is up to me to write for myself. With the ListManager, all I have to do is answer some simple questions. The program does the rest. (5)

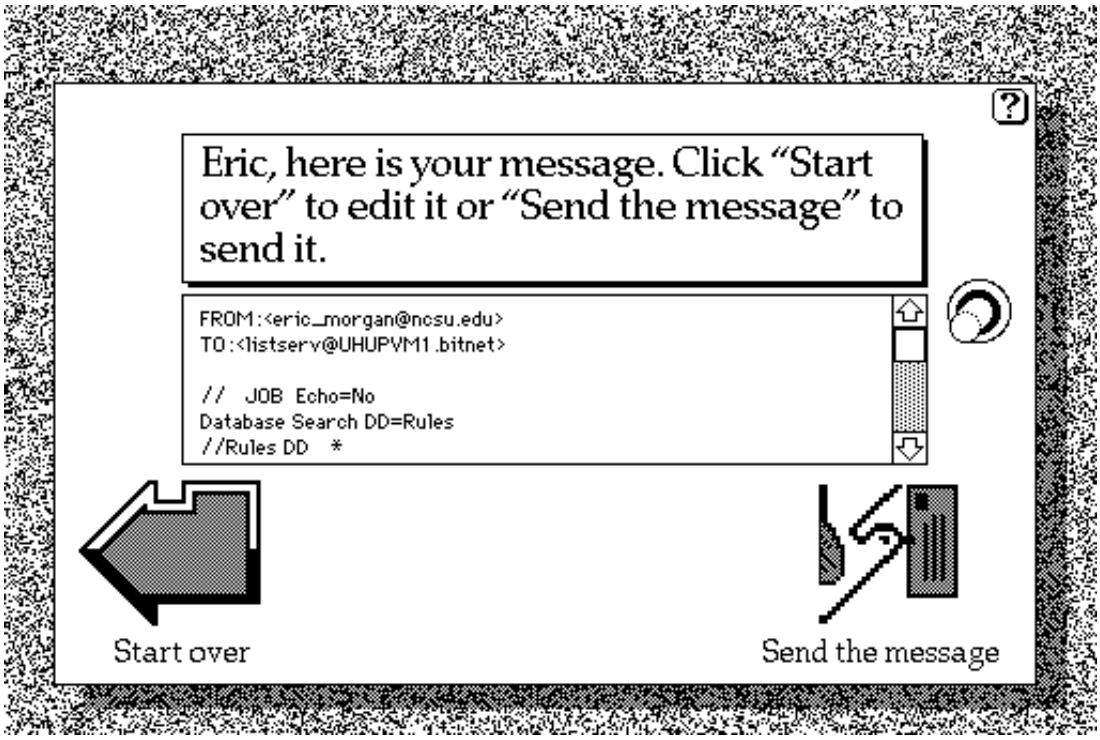

Figure 5. At the completion of the reference interview, the ListManager has created an SMTP mail message waiting to be sent by clicking the "Send the message" button with the help of XCMDs from the HyperCard TCP Toolkit.

After the message has been created, it is time to send it. The first step, like Mini-Atlas, is to resolve your mail server's IP or name address. (Step 1.) This is done with the TCPNameToAddr XCMD.

Second, we open a connection to the mail server on port 25 with the XCMD TCPActiveOpen. Port 25 is the standard port for SMTP mail transactions. (Step 2.)

Once a connection is established, then we begin to send the message line-by-line with the SendLine handler. (Steps 3., 4., and 5.) Unlike the Geographic Name Server, SMTP transaction prompts are numbers. Each number corresponds to a particular type of response. Numbers beginning with "5" are error messages. In general, numbers beginning with "2" or "3" mean there was no error and the dialog can continue. Therefore, the ListManager uses a modified version of the WaitForPeriod function called GetResponse. GetResponse takes one argument, a digit we want to receive from the mail server. If GetResponse "hears" the desired response, then the dialog continues. Otherwise, an error is returned and the connection is closed. (Steps 7. and 8.)

After the entire text of the SMTP message has been sent, the connection is closed with the Disconnect handler. (Steps 7. and 8.) Now all you have to do is sit back and wait for the LISTSERV software to send you mail. The reply will come to the email address you gave to the ListManager. Therefore, it doesn't matter what email program you use.

Summary

This article listed the necessary tools and outlined an algorithm for implementing TCP/IP communications with HyperCard. Two examples were then illustrated. First, Mini-Atlas queries the Geographic Name Server and reformats the results into human-readable form including placing them on a world map. Second, ListManager uses a reference interview model to create an SMTP message any LISTSERV program can understand. The resulting message is then sent to a LISTSERV program where it will be analyzed and acted upon.

In both of these examples there were two parts: the communications process and the information analysis process. This article detailed the communications process because it was common to both applications. The communications process described here hopefully has sparked your interest in creating your own information tools. For example, using a slightly modified version of the scripts in the Appendices, I have created a tiny stack that logs on to DIALOG, sends a previously saved search strategy, saves the results to another text file, and logs off. The program is only 38 K in size and it took me a couple of hours to write! With the scripts described in this article it would probably be possible to create WAIS, Gopher, or World Wide Web clients. These techniques have already been used to create applications that read UseNet news and

implement FTP transfers.

The hard part of all these applications is not the communications process. The hard part is creating the messages or analyzing the received data. More effort when into translating the output of the Geographic Name Server into human-readable form than went into the communications process. More effort went into learning the in's and out's of the SMTP protocol and the creation of an SMTP message than went into sending the message after it was created. This is what librarianship, in my opinion, is all about. We are here to help people with their information needs. Our profession attempts to understand the nature of information and how it is retrieved, organized, stored, and disseminated. With the advent of electronic information librarianship has become more complex and there is a need for at least a few librarians to know about this electronic information. The scripts described here exemplify one librarian's attempt to gain control over the electronic information and present it to patrons and other librarians in a usable form.

Notes:

1. I have found the TCP Toolkit at the following anonymous FTP sites:

   - akiu.gw.tohoku.ac.jp (130.34.8.9) /pub/mac/comm/MacTCP/mactcp-toolkit-1-0.hqx
   - ugle.unit.no (129.241.1.97) /pub/mac/mactcp-toolkit-1-0.hqx
   - ugle.unit.no (129.241.1.97) /pub/mac/mactcp-toolkit-1-0.hqx
   - src.doc.ic.ac.uk (146.169.2.1) /mac/mactcp-toolkit-1-0.hqx

   Alternatively, you can telnet to an Archie server (archie.mcgill.ca, archie.sura.net, archie.ans.net, archie.unl.edu, archie.rutgers.edu, archie.funet.fi, archie.au, archie.doc.ic.ac.uk, or cs.huji.ac.il), logon as "archie", and issue the command "prog mactcp" to find other locations.

2. Each computer on the Internet has a unique number. These numbers are stored in distrubuted databases and administered by computers called name servers. When you resolve an Internet name or address you are querying a name server for a remote computer's unique number, its address.
3. A port is analogous to an telephone extension. When you call a particular telephone number you may want to talk to someone on a different extension. The use of ports is the same idea.
4. Besides Boolean, adjacency, and date range queries, the LISTSERV software can do soundex searches. The ListManager specifies a soundex search with the "sounds like" qualifier. This gets passed on to the LISTSERV where, somehow, terms are analyzed phonetically. Don't ask me how it does it, it just does.
5. For more information concerning the construction of SMTP mail messages read RFC 821,Simple Mail Transfer Protocol by Postel. It is available via anonymous FTP from nic.ddn.mil as /rfc/rfc821.txt. For a listing of valid LISTSERV commands and information concerning the construction of LISTSERV database queries, send the commands get info and info database, respectively, to any LISTSERV program.

Appendix A: HyperTalk scipts for Mini-Atlas

```
--------------------------------------
-- query the Geographic Name Server --
--------------------------------------
on mouseUp
  global connectionID

  lock screen
  set cursor to busy

  -- clear the terminal screen, a place to old the results of our query
  put empty into card field "theTerminal"

  -- resolve the name
  prompt 1, "Resolving address for martini.eecs.umich.edu"   -- this is a Rinaldi XCMD
  set cursor to busy
  put item 1 of TCPNameToAddr("martini.eecs.umich.edu",,) into theIPAddress
  if theIPAddress contains "*" then

    -- an error occured in resolving the address
    prompt 3
    answer "There was an error resolving the address for martini.eecs.umich.edu." &
    return & return & theIPAddress
    exit mouseUp
```

```
    end if

    -- open the connection
    prompt 2, "Opening a connection to martini.eecs.umich.edu."
    set cursor to busy
    put TCPActiveOpen(theIPAddress, 3000, 0) into connectionID   -- the server lives on port 3000
    prompt 2, "Waiting for the period prompt."
    if WaitForPeriod() is "false" then exit mouseUp

    -- send the query
    put line 1 of card field "theQuery" into theQuery
    prompt 2, "Getting response(s) and waiting for the period prompt"
    set cursor to busy
    sendLine theQuery
    if WaitForPeriod() is "false" then exit mouseUp

    -- clean up
    sendLine "Quit"
    Disconnect
    prompt 3

    -- notify operator
    answer "The retrieval is complete." & return & return &
    "Do you want the results formatted now?" with "No" or "Format"
    if it is "Format" then
      send mouseUp to card button "Format"
    end if

end mouseUp

-------------------------
-- send a line of text --
-------------------------
on sendLine theText
  global connectionID

  -- send the text to the connection
  if TCPState (connectionID) is "established" then
    TCPSend connectionID, theText & return & linefeed
  end if

  -- display the text in the terminal window
  put theText & return after card field "theTerminal"

end sendLine


--------------------------------
-- wait for the period prompt --
--------------------------------
function WaitForPeriod
  global connectionID

  -- wait for the period (".") prompt
  repeat forever
    repeat 25 times
      put TCPRecvUpTo( connectionID, linefeed, 60, empty ) into theResponse
      set cursor to busy
      if theResponse is not empty then exit repeat
    end repeat

    -- strip the linefeed character
    delete the last character of theResponse

    -- display the response
    put theResponse after card field "theTerminal"

    -- analyse the response
    put the first character of theResponse into theCharacter
    if theCharacter is "." then

      -- got the period prompt
      put "true" into theReturnCode
      exit repeat

    end if
```

```
  end repeat

  -- return from the function
  return theReturnCode

end WaitForPeriod


--------------------------
-- close the connection --
--------------------------
on Disconnect
  global connectionID

  -- release the connection
  TCPClose connectionID
  TCPRelease connectionID

  -- dismiss the notification
  prompt 3

end disconnect
```

Appendix B: Hypertalk scripts for the ListManager

```
----------------------
-- send the message --
----------------------
on sendTheMessage
  global theMailServer, connectionID, theWholeMessage

  -- resolve the name
  put item 1 of TCPNameToAddr(theMailServer,,) into theIPAddress
  prompt 1, "Resolving address for " & theMailServer
  if theIPAddress contains "*" then

    -- an error occured in resolving the address
    prompt 3
    answer "There was an error resolving " & theMailServer & "." & return & return & theIPAddress
    hide card field "theTerminal"
    exit sendTheMessage
  end if

  -- open a connection
  put TCPActiveOpen(theIPAddress, 25, 0) into connectionID   -- port 25 is the SMTP socket
  prompt 2, "Opening a connection to " & theMailServer
  if getResponse("2") is "false" then exit sendTheMessage

  -- send the HELO message
  put line 1 of theWholeMessage into theHELOCommand
  sendLine theHELOCommand
  if getResponse("2") is "false" then exit sendTheMessage

  -- send the MAIL message
  put line 2 of theWholeMessage into theMAILCommand
  sendLine theMAILCommand
  if getResponse("2") is "false" then exit sendTheMessage

  -- send the RCPT message
  put line 3 of theWholeMessage into theRCPTCommand
  sendLine theRCPTCommand
  if getResponse("2") is "false" then exit sendTheMessage

  -- send the DATA message
  put line 4 of theWholeMessage into theDATACommand
  sendLine theDATACommand
  if getResponse("3") is "false" then exit sendTheMessage

  -- send the message
```

```
  put the number of lines of theWholeMessage into theNumberOfLines
  repeat with i = 5 to theNumberOfLines
    put line i of theWholeMessage into theText
    sendLIne theText
    idle
  end repeat

  -- clean up
  prompt 3
  disconnect
  hide card field "theTerminal"

end sendTheMessage


-------------------------------------------
-- get and analyse the server's  response --
-------------------------------------------
function getResponse aValidResponse
  global connectionID

  -- wait for a response
  repeat 25 times
    put TCPRecvUpTo( connectionID, linefeed, 60, empty ) into theResponse
    if theResponse is not empty then exit repeat
  end repeat

  -- analyse the response
  put the first character of theResponse into theCharacter
  if theCharacter is aValidResponse then

    -- strip the linefeed character
    delete the last character of theResponse

    -- display the response
    put return & theResponse after card field "theTerminal"

    return true

  else if theCharacter is "5" then

    -- an error occured
    prompt 3
    delete the last character of theResponse
    answer "There was an error." & return & return & theResponse
    TCPRelease connectionID

    return false

  else if theResponse is empty then

    -- there was no response
    prompt 3
    hide card field "theTerminal"
    answer "The remote computer is not responding."
    TCPRelease connectionID

    return false

  else

    -- an unknown error occured
    prompt 3
    answer "An unknown error occurred. Call Eric."
    TCPRelease connectionID

    return false

  end if

end getResponse
```

Creator: Eric Lease Morgan <eric_morgan@infomotions.com>
Source: Originally published in Information Technology and Libraries 11(4):421-432, December 1992.
Date created: 1992-12-21
Date updated: 2004-12-13
Subject(s): TCP/IP (Transmission Control Protocol/Internet Protocol); HyperCard; articles;
URL: http://infomotions.com/musings/tcp-communications/